Simulation and Monte Carlo

With applications in finance and MCMC

J. S. Dagpunar
School of Mathematics
University of Edinburgh, UK

John Wiley & Sons, Ltd

Simulation and Monte Carlo

Simulation and Monte Carlo

With applications in finance and MCMC

J. S. Dagpunar
School of Mathematics
University of Edinburgh, UK

John Wiley & Sons, Ltd

Copyright © 2007 John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wiley.com

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP, UK, without the permission in writing of the Publisher. Requests to the Publisher should be addressed to the Permissions Department, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, or emailed to permreq@wiley.co.uk, or faxed to (+44) 1243 770620.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The Publisher is not associated with any product or vendor mentioned in this book.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, ONT, Canada L5R 4J3

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging in Publication Data

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-0-470-85494-5 (HB) 978-0-470-85495-2 (PB)
ISBN-10: 0-470-85494-4 (HB) 0-470-85495-2 (PB)

Typeset in 10/12pt Times by Integra Software Services Pvt. Ltd, Pondicherry, India
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire
This book is printed on acid-free paper responsibly manufactured from sustainable forestry
in which at least two trees are planted for each one used for paper production.

To the memory of Jim Turner, Veterinary surgeon, 1916–2006

Contents

Preface xi

Glossary xiii

1 Introduction to simulation and Monte Carlo 1
- 1.1 Evaluating a definite integral 2
- 1.2 Monte Carlo is integral estimation 4
- 1.3 An example 5
- 1.4 A simulation using Maple 7
- 1.5 Problems 13

2 Uniform random numbers 17
- 2.1 Linear congruential generators 18
  - 2.1.1 Mixed linear congruential generators 18
  - 2.1.2 Multiplicative linear congruential generators 22
- 2.2 Theoretical tests for random numbers 25
  - 2.2.1 Problems of increasing dimension 26
- 2.3 Shuffled generator 28
- 2.4 Empirical tests 29
  - 2.4.1 Frequency test 29
  - 2.4.2 Serial test 30
  - 2.4.3 Other empirical tests 30
- 2.5 Combinations of generators 31
- 2.6 The seed(s) in a random number generator 32
- 2.7 Problems 32

3 General methods for generating random variates 37
- 3.1 Inversion of the cumulative distribution function 37
- 3.2 Envelope rejection 40
- 3.3 Ratio of uniforms method 44
- 3.4 Adaptive rejection sampling 48
- 3.5 Problems 52

4 Generation of variates from standard distributions 59
- 4.1 Standard normal distribution 59
  - 4.1.1 Box–Müller method 59
  - 4.1.2 An improved envelope rejection method 61
- 4.2 Lognormal distribution 62

4.3 Bivariate normal density 63
4.4 Gamma distribution 64
4.4.1 Cheng's log-logistic method 65
4.5 Beta distribution 67
4.5.1 Beta log-logistic method 67
4.6 Chi-squared distribution 69
4.7 Student's t distribution 69
4.8 Generalized inverse Gaussian distribution 71
4.9 Poisson distribution 73
4.10 Binomial distribution 74
4.11 Negative binomial distribution 74
4.12 Problems 75

5 Variance reduction 79
5.1 Antithetic variates 79
5.2 Importance sampling 82
5.2.1 Exceedance probabilities for sums of i.i.d. random variables 86
5.3 Stratified sampling 89
5.3.1 A stratification example 92
5.3.2 Post stratification 96
5.4 Control variates 98
5.5 Conditional Monte Carlo 101
5.6 Problems 103

6 Simulation and finance 107
6.1 Brownian motion 108
6.2 Asset price movements 109
6.3 Pricing simple derivatives and options 111
6.3.1 European call 113
6.3.2 European put 114
6.3.3 Continuous income 115
6.3.4 Delta hedging 115
6.3.5 Discrete hedging 116
6.4 Asian options 118
6.4.1 Naive simulation 118
6.4.2 Importance and stratified version 119
6.5 Basket options 123
6.6 Stochastic volatility 126
6.7 Problems 130

7 Discrete event simulation 135
7.1 Poisson process 136
7.2 Time-dependent Poisson process 140
7.3 Poisson processes in the plane 141
7.4 Markov chains 142
7.4.1 Discrete-time Markov chains 142
7.4.2 Continuous-time Markov chains 143

7.5 Regenerative analysis 144
7.6 Simulating a G/G/1 queueing system using the three-phase method 146
7.7 Simulating a hospital ward 149
7.8 Problems 151

8 Markov chain Monte Carlo 157
8.1 Bayesian statistics 157
8.2 Markov chains and the Metropolis–Hastings (MH) algorithm 159
8.3 Reliability inference using an independence sampler 163
8.4 Single component Metropolis–Hastings and Gibbs sampling 165
8.4.1 Estimating multiple failure rates 167
8.4.2 Capture–recapture 171
8.4.3 Minimal repair 172
8.5 Other aspects of Gibbs sampling 176
8.5.1 Slice sampling 176
8.5.2 Completions 178
8.6 Problems 179

9 Solutions 187
9.1 Solutions 1 187
9.2 Solutions 2 187
9.3 Solutions 3 190
9.4 Solutions 4 191
9.5 Solutions 5 195
9.6 Solutions 6 196
9.7 Solutions 7 202
9.8 Solutions 8 205

Appendix 1: Solutions to problems in Chapter 1 209

Appendix 2: Random number generators 227

Appendix 3: Computations of acceptance probabilities 229

Appendix 4: Random variate generators (standard distributions) 233

Appendix 5: Variance reduction 239

Appendix 6: Simulation and finance 249

Appendix 7: Discrete event simulation 283

Appendix 8: Markov chain Monte Carlo 299

References 325

Index 329

Preface

This book provides an introduction to the theory and practice of Monte Carlo and Simulation methods. It arises from a 20 hour course given simultaneously to two groups of students. The first are final year Honours students in the School of Mathematics at the University of Edinburgh and the second are students from Heriot Watt and Edinburgh Universities taking the MSc in Financial Mathematics.

The intention is that this be a practical book that encourages readers to write and experiment with actual simulation models. The choice of programming environment, Maple, may seem strange, perhaps even perverse. It arises from the fact that at Edinburgh all mathematics students are conversant with it from year 1. I believe this is true of many other mathematics departments. The disadvantage of slow numerical processing in Maple is neutralized by the wide range of probabilistic, statistical, plotting, and list processing functions available. A large number of specially written Maple procedures are available on the website accompanying this book (www.wiley.com/go/dagpunar_simulation). They are also listed in the Appendices.[1]

The content of the book falls broadly into two halves, with Chapters 1 to 5 mostly covering the theory and probabilistic aspects, while Chapters 6 to 8 cover three application areas. Chapter 1 gives a brief overview of the breadth of simulation. All problems at the end of this chapter involve the writing of Maple procedures, and full solutions are given in Appendix 1. Chapter 2 concerns the generation and assessment of pseudo-random numbers. Chapter 3 discusses three main approaches to the sampling (generation) of random variates from distributions. These are: inversion of the distribution function, the envelope rejection method, and the ratio of uniforms method. It is recognized that many other methods are available, but these three seem to be the most frequently used, and they have the advantage of leading to easily programmed algorithms. Readers interested in the many other methods are directed to the excellent book by Devroye (1986) or an earlier book of mine (Dagpunar, 1988a). Two short Maple procedures in Appendix 3 allow readers to quickly ascertain the efficiency of rejection type algorithms. Chapter 4 deals with the generation of variates from standard distributions. The emphasis is on short, easily implemented algorithms. Where such an algorithm appears to be faster than the corresponding one in the Maple statistics package, I have given a listing in Appendix 4. Taken together, I hope that Chapters 3 and 4 enable readers to understand how the generators available in various packages work and how to write algorithms for distributions that either do not appear in such packages or appear to be slow in execution. Chapter 5 introduces variance reduction methods. Without these, many simulations are incapable of giving precise estimates within a reasonable amount of processing time. Again, the emphasis is on an empirical approach and readers can use the procedures in

[1] The programs are provided for information only and may not be suitable for all purposes. Neither the author nor the publisher is liable, to the fullest extent permitted by law, for any failure of the programs to meet the user's requirements or for any inaccuracies or defects in the programs.

Appendix 5 to illustrate the efficacy of the various designs, including importance and stratified sampling.

Chapters 6 and 8, on financial mathematics and Markov chain Monte Carlo methods respectively, would not have been written 10 years ago. Their inclusion is a result of the high-dimensional integrations to be found in the pricing of exotic derivatives and in Bayesian estimation. In a stroke this has caused a renaissance in simulation. In Chapter 6, I have been influenced by the work of Glasserman (2004), particularly his work combining importance and stratified sampling. I hope in Sections 6.4.2 and 6.5 that I have provided a more direct and accessible way of deriving and applying such variance reduction methods to Asian and basket options. Another example of high-dimensional integrations arises in stochastic volatility and Section 6.6 exposes the tip of this iceberg. Serious financial engineers would not use Maple for simulations. Nevertheless, even with Maple, it is apparent from the numerical examples in Chapter 6 that accurate results can be obtained in a reasonable amount of time when effective variance reduction designs are employed. I also hope that Maple can be seen as an effective way of experimenting with various models, prior to the final construction of an efficient program in C++ or Java, say. The Maple facility to generate code in, say, C++ or Fortran is useful in this respect.

Chapter 7 introduces discrete event simulation, which is perhaps best known to operational researchers. It starts with methods of simulating various Markov processes, both in discrete and continuous time. It includes a discussion of the regenerative method of analysing autocorrelated simulation output. The simulation needs of the operational researcher, the financial engineer, and the Bayesian statistician overlap to a certain extent, but it is probably true to say that no single computing environment is ideal for all application fields. An operational researcher might progress from Chapter 7 to make use of one of the powerful purpose-built discrete event simulation languages such as Simscript II.5 or Witness. If so, I hope that the book provides a good grounding in the principles of simulation.

Chapter 8 deals with the other burgeoning area of simulation, namely Markov chain Monte Carlo and its use in Bayesian statistics. Here, I have been influenced by the works of Robert and Casella (2004) and Gilks *et al.* (1996). I have also included several examples from the reliability area since the repair and maintenance of systems is another area that interests me. Maple has been quite adequate for the examples discussed in this chapter. For larger hierarchical systems a purpose-built package such as BUGS is the answer.

There are problems at the end of each chapter and solutions are given to selected ones. A few harder problems have been designated accordingly. In the text and problems, numerical answers are frequently given to more significant figures than the data would warrant. This is done so that independent calculations may be compared with the ones appearing here.

I am indebted to Professor Alastair Gillespie, head of the School of Mathematics, Edinburgh University, for granting me sabbatical leave for the first semester of the 2005–2006 session. I should also like to acknowledge the several cohorts of simulation students that provided an incentive to write this book. Finally, my thanks to Angie for her encouragement and support, and for her forbearance when I was not there.

Glossary

beta (α, β)[1]	A random variable that is beta distributed with p.d.f. $f(x) = \Gamma(\alpha+\beta)\, x^{\alpha-1}(1-x)^{\beta-1}/[\Gamma(\alpha)\,\Gamma(\beta)]$, $1 \geq x \geq 0$, where $\alpha > 0$, $\beta > 0$.
binomial (n, p)	A binomially distributed random variable.
$\{B(t), t \geq 0\}$	Standard Brownian motion.
c.d.f.	Cumulative distribution function.
$\mathbf{C}'$	The transpose of a matrix $\mathbf{C}$.
$\mathrm{Cov}_f(X, Y)$	The covariance of X and Y where $f(x, y)$ is the joint p.d.f./p.m.f. of X and Y (the subscript is often dropped).
e.s.e.	Estimated standard error.
Exp (λ)	A r.v. that has the p.d.f. $f(x) = \lambda \mathrm{e}^{-\lambda x}$, $x \geq 0$, where $\lambda > 0$.
$E_f(X)$	The expectation of a random variable X that has the p.d.f. or p.m.f. f (the subscript is often dropped).
$f_X(x)$	The p.d.f. or p.m.f. of a random variable X (the subscript is often dropped).
$F_X(x)$	The c.d.f. of a random variable X.
$\overline{F}_X(x)$	Complementary cumulative distribution function $[= 1 - F_X(x)]$.
gamma (α, λ)	A gamma distributed r.v. with the p.d.f. $f(x) = \lambda^\alpha x^{\alpha-1}\mathrm{e}^{-\lambda x}/\Gamma(\alpha)$, $x \geq 0$, where $\alpha > 0$, $\lambda > 0$.
gig (λ, ψ, χ)	A r.v. distributed as a generalized inverse Gaussian distribution with the p.d.f. $f(x) \propto x^{\lambda-1}\exp\left(-\frac{1}{2}[\psi x + \chi/x]\right)$, $x \geq 0$.
i.i.d.	Identically and independently distributed.
negbinom (k, p)	A negative binomial r.v. with the p.d.f. $f(x) \propto p^x(1-p)^k$, $x = 0, 1, 2, \ldots$, where $0 < p < 1$.
$N\left(\mu, \sigma^2\right)$	A normal r.v. with expectation μ and variance σ^2, or the density itself.
$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A vector r.v. $\mathbf{X}$ distributed as multivariate normal with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
p.d.f.	Probability density function.
p.m.f.	Probability mass function.
Poisson (μ)	A r.v. distributed as a Poisson with the p.m.f. $f(x) = \mu^x \mathrm{e}^{-\mu x}/x!$, $x = 0, 1, \ldots$, where $\mu > 0$.
$P(X < x)$	Probability that the random variable X is less than x.
$P(X = x)$	Probability that a (discrete) random variable equals x.
r.v.	Random variable.
s.e.	Standard error.

[1] This can also refer to the distribution itself. This applies to all corresponding random variable names in this list.

support (f)	$\{x : x \in \mathbb{R}, f(x) \neq 0\}$.
$U(0, 1)$	A continuous r.v. that is uniformly distributed in the interval (0, 1).
$\mathrm{Var}_f(X)$	The variance of a random variable X that has the p.d.f. or p.m.f f (the subscript is often dropped).
v.r.r.	Variance reduction ratio.
Weibull (α, λ)	A Weibull distributed random variable with the p.d.f. $f(x) = \alpha\lambda^{\alpha}x^{\alpha-1}\mathrm{e}^{-(\lambda x)^{\alpha}}/\Gamma(\alpha)$, $x \geq 0$, where $\alpha > 0, \lambda > 0$.
x^+	$\max(x, 0)$.
$\sigma_f(X)$	The standard deviation of a random variable X that has the p.d.f. or p.m.f. f (the subscript is often dropped).
$\phi(z)$	The p.d.f. for the standard normal.
$\Phi(z)$	The c.d.f. for the standard normal.
χ^2_n	An r.v. distributed as chi-squared with n degrees of freedom, $n = 1, 2, \ldots$. Therefore, $\chi^2_n = \mathrm{gamma}(n/2, 1/2)$.
1_P	Equals 1 if P is true, otherwise equals 0.
$\sim$	'Is distributed as'. For example, $X \sim \mathrm{Poisson}(\mu)$ indicates that X has a Poisson distribution.
$:= y$	In Maple or in pseudo-code this means 'becomes equal to'. The value of the expression to the right of := is assigned to the variable or parameter to the left of :=.

1

Introduction to simulation and Monte Carlo

A *simulation* is an experiment, usually conducted on a computer, involving the use of *random numbers*. A random number stream is a sequence of statistically independent random variables uniformly distributed in the interval [0,1). Examples of situations where simulation has proved useful include:

(i) modelling the flow of patients through a hospital;

(ii) modelling the evolution of an epidemic over space and time;

(iii) testing a statistical hypothesis;

(iv) pricing an option (derivative) on a financial asset.

A feature common to all these examples is that it is difficult to use purely analytical methods to either model the real-life situation [examples (i) and (ii)] or to solve the underlying mathematical problems [examples (iii) and (iv)]. In examples (i) and (ii) the systems are stochastic, there may be complex interaction between resources and certain parts of the system, and the difficulty may be compounded by the requirement to find an 'optimal' strategy. In example (iii), having obtained data from a statistical investigation, the numerical value of some test statistic is calculated, but the distribution of such a statistic under a null hypothesis may be impossibly difficult to derive. In example (iv), it transpires that the problem often reduces to evaluating a multiple integral that is impossible to solve by analytical or conventional numerical methods. However, such integrals can be estimated by *Monte Carlo* methods. Dating from the 1940s, these methods were used to evaluate definite multiple integrals in mathematical physics. There is now a resurgence of interest in such methods, particularly in finance and statistical inference.

In general, simulation may be appropriate when there is a problem that is too difficult to solve analytically. In a simulation a controlled *sampling experiment* is conducted on a computer using random numbers. *Statistics* arising from the sampling experiments (examples are sample mean, sample proportion) are used to *estimate* some parameters of interest in the original problem, system, or population.

Simulation and Monte Carlo: With applications in finance and MCMC J. S. Dagpunar
© 2007 John Wiley & Sons, Ltd

Since simulations provide an *estimate* of a parameter of interest, there is always some error, and so a quantification of the *precision* is essential, and forms an important part of the design and analysis of the experiment.

1.1 Evaluating a definite integral

Suppose we wish to evaluate the integral

$$I_\alpha = \int_0^\infty x^{\alpha-1} \mathrm{e}^{-x} \mathrm{d}x \tag{1.1}$$

for a specific value of $\alpha > 0$. Consider a random variable X having the probability density function (p.d.f.) f on support $[0, \infty)$ where

$$f(x) = \mathrm{e}^{-x}.$$

Then from the definition of the expectation of a function of a random variable Equation (1.1) leads to

$$I_\alpha = E_f\left(X^{\alpha-1}\right).$$

It follows that a (statistical) estimate of I_α may be obtained by conducting the following controlled sampling experiment. Draw a random sample of observations $X_1, X_2, \ldots, X_n$ from the probability density function f. Construct the statistic

$$\widehat{I}_\alpha = \frac{1}{n} \sum_{i=1}^{n} X_i^{\alpha-1}. \tag{1.2}$$

Then $\widehat{I}_\alpha$ is an unbiased estimator of I_α and assuming the $\{X_i\}$ are independent, the variance of $\widehat{I}_\alpha$ is given by

$$\mathrm{Var}_f\left(\widehat{I}_\alpha\right) = \frac{\mathrm{Var}_f\left(X^{\alpha-1}\right)}{n}.$$

Thus, the standard deviation of the sampling distribution of the statistic $\widehat{I}_\alpha$ is

$$\sigma_f\left(\widehat{I}_\alpha\right) = \frac{\sigma_f\left(X^{\alpha-1}\right)}{\sqrt{n}}.$$

This is the *standard error* (s.e.) of the statistic and varies as $1/\sqrt{n}$. Therefore, to change the standard error by a factor of K, say, requires the sample size to change by a factor of $1/K^2$. Thus, extra precision comes at a disproportionate extra cost.

By way of a numerical example let us estimate the value of the definite integral

$$I_{1.9} = \int_0^\infty x^{0.9} \mathrm{e}^{-x} \mathrm{d}x.$$

Firstly, we need to know how to generate values from the probability density function $f(x) = \mathrm{e}^{-x}$. It will be seen in Chapter 3 that this is done by setting

$$X_i = -\ln R_i \tag{1.3}$$

where $\{R_i : i = 0, 1, \ldots\}$ is a *random number stream* with $R_i \sim U(0, 1)$. From a built-in calculator function the following random numbers were obtained:

$$R_1 = 0.0078, \quad R_2 = 0.9352, \quad R_3 = 0.1080, \quad R_4 = 0.0063.$$

Using these in Equations (1.3) and (1.2) gives $\widehat{I}_{1.9} = 2.649$. In fact, the true answer to five significant figures (from tables of the gamma function) is $I_{1.9} = \Gamma(1.9) = 0.96177$. Therefore, the estimate is an awful one. This is not surprising since only four values were sampled. How large should the sample be in order to give a standard error of 0.001, say? To answer this we need to know the standard error of $\widehat{I}_{1.9}$ when $n = 4$. This is unknown as $\sigma_f\left(X^{0.9}\right)$ is unknown. However, the sample standard deviation of $X^{0.9}$ is s where

$$s = \sqrt{\frac{1}{4-1}\left(\sum_{i=1}^{4}\left[x_i^{0.9}\right]^2 - \left\{\sum_{i=1}^{4} x_i^{0.9}\right\}^2 /4\right)} = 1.992 \tag{1.4}$$

and $\{x_i : i = 1, \ldots, 4\}$ is the set of four values sampled from f. Therefore, the estimated standard error (e.s.e.) is $s/\sqrt{4} = 0.9959$. In order to reduce the standard error to 0.001, an approximately 996-fold reduction in the standard error would be needed, or a sample of approximate size $4 \times 996^2 \approx 3.97 \times 10^6$. We learn from this that an uncritical design and analysis of a simulation will often lead to a vast consumption of computer time.

Is it possible to design the experiment in a more efficient way? The answer is 'yes'. Rewrite the integral as

$$I_{1.9} = \int_0^\infty x\mathrm{e}^{-x}\left(\frac{1}{x^{0.1}}\right)\mathrm{d}x. \tag{1.5}$$

There is a convenient method of generating variates $\{x\}$ from the probability density function

$$g(x) = x\mathrm{e}^{-x} \tag{1.6}$$

with support $[0, \infty)$. It is to take two random numbers $R_1, R_2, \sim U(0, 1)$ and set

$$X = -\ln R_1 R_2.$$

Given this, Equation (1.5) can be written as

$$I_{1.9} = E_g\left(\frac{1}{X^{0.1}}\right)$$

where X has the density of Equation (1.6). Given the same four random numbers, two random values (variates) can be generated from Equation (1.6). They are $x_1 = -\ln R_1 R_2 = 4.9206$ and $x_2 = -\ln R_3 R_4 = 7.2928$. Therefore,

$$\widehat{I_{1.9}} = \frac{1}{2}(4.9206^{-0.1} + 7.2928^{-0.1}) = 0.8363. \tag{1.7}$$

This is a great improvement on the previous estimate. A theoretical analysis shows that $\sigma_g\left(\widehat{I}_{1.9}(X_1,\ldots,X_n)\right)$ is much smaller than $\sigma_f\left(\widehat{I}_{1.9}(X_1,\ldots,X_n)\right)$, the reason being that $\sigma_g\left(1/X^{0.1}\right) << \sigma_f\left(X^{0.9}\right)$.

Now try to estimate $I_{1.99}$ using both methods with the same random numbers as before. It is found that when averaging $1/X^{0.01}$ sampled from g, $\widehat{I}_{1.99} = 0.98226$, which is very close to the true value, $I_{1.99} = 0.99581$. This is not the case when averaging $X^{0.99}$ sampled from f.

This simple change in the details of the sampling experiment is an example of a *variance reduction* technique. In this case it is known as *importance sampling*, which is explored in Chapter 5.

1.2 Monte Carlo is integral estimation

How is it that the Monte Carlo method evolved from its rather specialized use in integral evaluation to its current status as a modelling aid is used to understand the behaviour of complex stochastic systems? Let us take the example of some type of queue, for example one encountered in a production process. We might be interested in the expectation of the total amount of time spent waiting by the first n 'customers', given some initial state S_0. In this case we wish to find

$$E_f\{W_1+\cdots+W_n\,|S_0\}$$

where f is the multivariate probability density of $W_1,\ldots,W_n$ given S_0. For most systems of any practical interest it will be difficult, probably impossible, to write down the density f. At first sight this might appear to rule out the idea of generating several, say m, realizations $\left\{\left\{w_1^{(i)},\ldots,w_n^{(i)}\right\}: i=1,\ldots,m\right\}$. However, it should not be too difficult to generate a realization of the waiting time W_1 of the first customer. Given this, the known structure of the system is then used to generate a realization of W_2 given a knowledge of the state of the system at all times up to the departure of customer 1. Note that it is often much easier to generate a *realization* of W_2 (given W_1 and the state of the system up to the departure of customer 1) than it is to write down the conditional *distribution* of W_2 given W_1. This is because the value assumed by W_2 can be obtained by breaking down the evolution of the system between the departures of customers 1 and 2 into easy stages. In this way it is possible to obtain realizations of values of $W_1,\ldots,W_n$.

The power of Monte Carlo lies in the ability to estimate the value of any definite integral, no matter how complex the integrand. For example, it is just as easy to estimate $E_f\{\text{Max}(W_1,\ldots,W_n)\,|S_0\}$ as it is to estimate $E_f\{W_1+\cdots+W_n\,|S_0\}$. Here is another example. Suppose we wish to estimate the expectation of the time average of queue length in $[0,T]$. Monte Carlo is used to estimate $E_f\left\{(1/T)\int_0^T Q(t)\,\mathrm{d}t\right\}$, where $\{Q(t), t\geq 0\}$ is a stochastic process giving the queue length and f is the probability density for the paths $\{Q(t), t\geq 0\}$. Again, a realization of $\{Q(t), T\geq t\geq 0\}$ is obtained by breaking the 'calculation' down into easy stages. In practice it may be necessary to discretize the time interval $[0,T]$ into a large number of small subintervals. A further example is given. Suppose there is a directed acyclic graph in which the arc lengths represent random costs

that are statistically dependent. We wish to find the probability that the shortest path through the network has a length (cost) that does not exceed x, say. Write this probability as $P(X < x)$. It can be seen that

$$P(X < x) = \int_{-\infty}^{\infty} f(t)\,(1 - 1_{x>t})\,\mathrm{d}t$$

where f is the probability density of the length of the shortest path and $1_{x>t} = 1$ if $x > t$; else $1_{x>t} = 0$. Since the probability can be expressed as an integral and since realizations of $1_{x>t}$ can be simulated by breaking down the calculation into easier parts using the structure of the network, the probability can be estimated using Monte Carlo. In fact, if 'minimum' is replaced by 'maximum' we have a familiar problem in project planning. This is the determination of the probability that the duration of a project does not exceed some specified value, when the individual activity durations are random and perhaps statistically dependent. Note that in all these examples the integration is over many variables and would be impossible by conventional numerical methods, even when the integrand can be written down.

The words 'Monte Carlo' and 'simulation' tend to be used interchangeably in the literature. Here a *simulation* is defined as a controlled experiment, usually carried out on a computer, that uses $U(0, 1)$ random numbers that are statistically independent. A *Monte Carlo* method is a method of estimating the value of an integral (or a sum) using the realized values from a simulation. It exploits the connection between an integral (or a sum) and the expectation of a function of a(some) random variable(s).

1.3 An example

Let us now examine how a Monte Carlo approach can be used in the following problem. A company owns K skips that can be hired out. During the nth day $(n = 1, 2, \ldots)$ Y_n people approach the company each wishing to rent a single skip. $Y_1, Y_2, \ldots$ are independent random variables having a Poisson distribution with mean λ. If skips are available they are let as 'new hires'; otherwise an individual takes his or her custom elsewhere. An individual hire may last for several days. In fact, the probability that a skip currently on hire to an individual is returned the next day is p. Skips are always returned at the beginning of a day. Let $X_n(K)$ denote the total number of skips on hire at the end of day n and let $H_n(K)$ be the number of new hires during day n. To simplify notation the dependence upon K will be dropped for the moment.

The problem is to find the optimal value of K. It is known that each new hire generates a fixed revenue of £c_f per skip and a variable revenue of £c_v per skip per day or part-day. The K skips have to be bought at the outset and have to be maintained irrespective of how many are on hire on a particular day. This cost is equivalent to £c_0 per skip per day.

Firstly, the stochastic process $\{X_n, n = 0, 1, \ldots\}$ is considered. Assuming skips are returned at the beginning of a day before hiring out,

$$\begin{aligned}
&Y_n = \text{Poisson}(\lambda) \quad (n = 1, 2, \ldots),\\
&X_{n+1} = \min\{K, \text{binomial}(X_n, 1 - p) + Y_{n+1}\} \quad (n = 0, 1, 2, \ldots),\\
&H_{n+1} = X_{n+1} - \text{binomial}(X_n, 1 - p) \quad (n = 0, 1, 2, \ldots).
\end{aligned}$$

Since X_{n+1} depends on X_n and Y_{n+1} only, and since Y_{n+1} is independent of $X_{n-1}, X_{n-2}, \ldots,$ it follows that $\{X_n, n = 0, 1, \ldots\}$ is a discrete-state, discrete-time, homogeneous Markov chain. The probability transition matrix is $\mathbf{P} = \left\{p_{ij} : i, j = 0, \ldots, K\right\}$ where $p_{ij} = P(X_{n+1} = j | X_n = i)$ for all n. Given that i skips are on hire at the end of day n the probability that r of them remain on hire at the beginning of day $n+1$ is the binomial probability $\binom{i}{r}(1-p)^r p^{i-r}$. The probability that there are $j-r$ people requesting new hires is the Poisson probability $\lambda^{j-r}\mathrm{e}^{-\lambda}/(j-r)!$ Therefore, for $0 \le i \le K$ and $0 \le j \le K-1$,

$$p_{ij} = \sum_{r=0}^{\min(i,j)} \binom{i}{r}(1-p)^r p^{i-r} \frac{\lambda^{j-r}\mathrm{e}^{-\lambda}}{(j-r)!}. \tag{1.8}$$

For the case $j = K$,

$$p_{iK} = 1 - \sum_{j=0}^{K-1} p_{ij}. \tag{1.9}$$

Since $p_{ij} > 0$ for all i and j, the Markov chain is ergodic. Therefore, $P(X_n = j) \to P(X = j) = \pi_j$, say, as $n \to \infty$. Similarly, $P(H_n = j) \to P(H = j)$ as $n \to \infty$. Let $\boldsymbol{\pi}$ denote the row vector $(\pi_0, \ldots, \pi_K)$. $\boldsymbol{\pi}$ is the *stationary distribution* of the chain $\{X_n, n = 0, 1, \ldots\}$. Suppose we wish to maximize the long-run average profit per day. Then a K is found that maximizes $Z(K)$ where

$$Z(K) = c_{\mathrm{f}} E(H[K]) + c_{\mathrm{v}} E(X[K]) - c_0 K \tag{1.10}$$

and where the dependence upon K has been reintroduced. Now

$$\begin{aligned} E(H[K]) &= \lim_{n\to\infty} \{E(H_n[K])\} \\ &= \lim_{n\to\infty} \{E(X_n[K]) - E(\text{binomial}(X_{n-1}[K], 1-p))\} \\ &= E(X[K]) - E(X[K])(1-p) \\ &= pE(X[K]) \end{aligned}$$

This last equation expresses the obvious fact that in the long run the average number of new hires per day must equal the average returns per day. Substituting back into Equation (1.10) gives

$$Z(K) = E(X[K])(pc_{\mathrm{f}} + c_{\mathrm{v}}) - c_0 K$$

The first difference of $Z(K)$ is

$$\begin{aligned} D(K) &= Z(K+1) - Z(K) \\ &= \{E(X[K+1]) - E(X[K])\}[pc_{\mathrm{f}} + c_{\mathrm{v}}] - c_0 \end{aligned}$$

for $K = 0, 1, \ldots$. It is obvious that increasing K by one will increase the expected number on hire, and it is reasonable to assume that $E(X[K+1]) - E(X[K])$ is decreasing in K. In that case $Z(K)$ will have a unique maximum.

To determine the optimal K we require $E(X[K])$ for successive integers K. If K is large this will involve considerable computation. Firstly, the probability transition matrix would have to be calculated using Equations (1.8) and (1.9). Then it is necessary to invert a $(K+1)\times(K+1)$ matrix to find the stationary distribution $\boldsymbol{\pi}$ for each K, and finally we must compute $E(X[K])$ for each K.

On the other hand, the following piece of *pseudo-code* will *simulate* $X_1(K),\ldots,X_n(K)$:

```
Input K, X_0(K), λ, p, n
x := X_0(K)
For i = 1, ..., n
  y := Poisson(λ)
  r := binomial(x, 1 − p)
  x := min(r + y, K)
  Output x
Next i
```

Now $E(X[K])=\sum_{x=0}^{K} x\pi_x$. Therefore, if $\{X_1(K),\ldots,X_n(K)\}$ is a sample from $\boldsymbol{\pi}$, an unbiased estimator of $E(X[K])$ is $(1/n)\sum_{i=1}^{n} X_i(K)$. There is a minor inconvenience in that unless $X_0(K)$ is selected randomly from $\boldsymbol{\pi}$, the stationary distribution, then $\{X_1(K),\ldots,X_n(K)\}$ will not be precisely from $\boldsymbol{\pi}$, and therefore the estimate will be slightly *biased*. However, this may be rectified by a *burn-in* period of b observations, say, followed by a further $n-b$ observations. We then estimate $E(X[K])$ using $[1/(n-b)]\sum_{i=b+1}^{n} X_i(K)$.

In terms of programming effort it is probably much easier to use this last Monte Carlo approach. However, it must be remembered that it gives an *estimate* while the method involving matrix inversion gives the *exact* value, subject to the usual numerical roundoff errors. Further, we may have to simulate for a considerable period of time. It is not necessarily the burn-in time that will be particularly lengthy. The fact that the members of $\{X_i(K), i=0,1,2,\ldots\}$ are *not independent* (they are *auto-correlated*) will necessitate long sample runs if a precise estimate is required.

Finding the optimal K involves determining $E(X[K+1])-E(X[K])$ with some precision. We can reduce the sampling variation in our estimate of this by inducing positive correlation between our estimates of $E(X[K+1])$ and $E(X[K])$. Therefore, we might consider making $Y_n[K]=Y_n[K+1]$ and making binomial $(X_n[K],1-p)$ positively correlated with binomial $(X_n[K+1],1-p)$. Such issues require careful experimental planning. The aim is that of variance reduction, as seen in Section 1.1.

1.4 A simulation using Maple

This section contains an almost exact reproduction of a Maple worksheet, 'skipexample.mws'. It explores the problem considered in Section 1.3. It can also be downloaded from the website accompanying the book.

It will now be shown how the algorithm for simulation of skip hires can be programmed as a Maple *procedure*. Before considering the procedure we will start with

a fresh worksheet by typing 'restart' and also load the statistics package by typing 'with(stats)'. These two lines of input plus the Maple generated response, '[anova, . . . , transform]' form an *execution group*. In the downloadable file this is delineated by an elongated left bracket, but this is not displayed here.

```
> restart;
  with (stats);[anova, describe, fit, importdata, random, statevalf,
                statplots, transform]
```

A Maple procedure is simply a function constructed by the programmer. In this case the name of the procedure is *skip* and the arguments of the function are λ, p, K, $x0$ and n, the number of days to be simulated. Subsequent calling of the procedure *skip* with selected values for these parameters will create a *sequence;* $hire[1], \ldots, hire[n]$ where $hire[i] = [i, x]$. In Maple terminology $hire[i]$ is itself a *list* of two items: i the day number and x the total number of skips on hire at the end of that day. Note how a list is enclosed by square brackets, while a sequence is not.

Each Maple procedure starts with the Maple prompt '>'. The procedure is written within an *execution group*. Each line of code is terminated by a semicolon. However, anything appearing after the '#' symbol is not executed. This allows programmer comments to be added. Use the 'shift' and 'return' keys to obtain a fresh line within the procedure. The procedure terminates with a semicolon and successful entry of the procedure results in the code being 'echoed' in blue type. The structure of this echoed code is highlighted by appropriate automatic indentation.

```
> skip: =proc (lambda, p, K, x0, n) local x, i, y, r, hire;
  randomize (5691443); # An arbitrary integer sets the 'seed'
  for the U(0, 1) random number generator.
  x:=x0; # x is the total number on hire, initially set to x0
  for i from 1 to n do;
    y:=stats[random, poisson[lambda]] (1, 'default',
    'inverse'); # Generates a random
  Poisson variate; 1 is the number of variates generated,
  'default' is MAPLE's default uniform generator, 'inverse'
  means that the variate is generated by inverting the
  cumulative distribution function [see Chapter 3]
    if x=0 then r:=0 else
  r:=stats[random, binomiald[x,1−p]] (1, 'default',
  'inverse') end if; # Generates a random binomial variate
    x:=min (r+y, K); # Updates the total number on hire
    hire[i]: =[i, x];
  end do;
  seq(hire[i], i=1 . . n); #Assigns the sequence hire[1], . . . ,
  hire[n] to procedure 'SKIP';
  end proc;
```

skip := **proc**(λ, p, K, $x0$, n)
local *x, i, y, r, hire*;
    randomize(5691443);

```
    x := x0;
    for i to n do
        y := stats[random, poisson[λ]](1, 'default', 'inverse');
        if x = 0 then r := 0
        else r :=stats[random, binomiald[x, 1 − p]](1, 'default', 'inverse')
        end if;
        x := min(r + y, K);
        hire[i] := [i, x]
    end do;
    seq(hire[i], i = 1..n)
end proc
```

The 'echoed' code is now examined. *Local* variables are those whose values cannot be transmitted to and from the procedure. The 'randomize' statement uses an arbitrarily chosen integer to set a *seed* for a $U(0, 1)$ random number generator within Maple. More will be said about such generators in Chapter 2. Initially, the total number of skips $x0$ on hire in the current day (day 0) is assigned to the variable x. This is followed by a 'for i to…do…end do' loop. The statements within this loop are executed for $i = 1, \ldots, n$. Maple contains a 'stats' package and 'random' is a subpackage of this. Note the use of the function 'poisson' within this subpackage. The generated Poisson variate is assigned to the variable y. Following this is an example of a conditional 'if…then…else…endif' statement. If the total number of skips on hire on the previous day is zero, then the number remaining on hire today (r) must be zero; otherwise r equals a random binomial variate with parameters x and $1 - p$. Following this, the value of the variable x is updated. Then the list $[i, x]$ is assigned to the variable $hire[i]$. The last executable statement forms a sequence $hire[1], \ldots, hire[n]$. Maple procedures use the convention that the result of the last statement that is executed is assigned to the name of the procedure. So when *skip* is subsequently called it will output a random realization of this sequence.

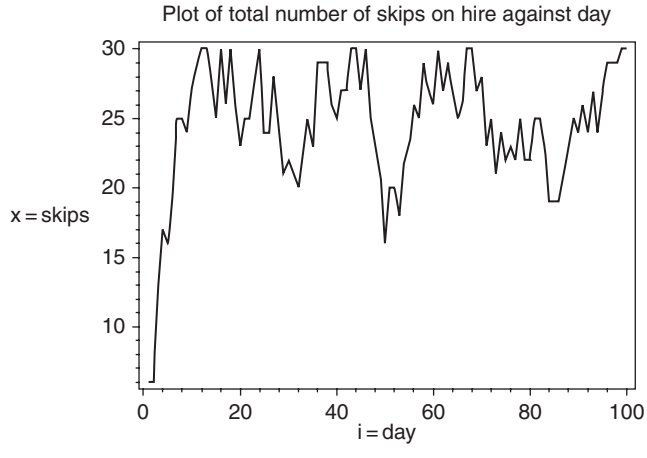
Some results will now be obtained when $\lambda = 5$ per day, $p = 0.2, K = 30, x0 = 0$, and $n = 100$ days. Calling 'skip' and assigning the results to a variable *res* gives the sequence *res*.

```
> res :=skip (5, 0.2, 30, 0, 100);
```

res := [1, 6], [2, 6], [3, 13], [4, 17], [5, 16], [6, 19], [7, 25], [8, 25], [9, 24], [10, 27], [11, 29], [12, 30], [13, 30], [14, 28], [15, 25], [16, 30], [17, 26], [18, 30], [19, 26], [20, 23], [21, 25], [22, 25], [23, 28], [24, 30], [25, 24], [26, 24], [27, 28], [28, 24], [29, 21], [30, 22], [31, 21], [32, 20], [33, 22], [34, 25], [35, 23], [36, 29], [37, 29], [38, 29], [39, 26], [40, 25], [41, 27], [42, 27], [43, 30], [44, 30], [45, 27], [46, 30], [47, 25], [48, 23], [49, 21], [50, 16], [51, 20], [52, 20], [53, 18], [54, 22], [55, 23], [56, 26], [57, 25], [58, 29], [59, 27], [60, 26], [61, 30], [62, 27], [63, 29], [64, 27], [65, 25], [66, 26], [67, 30], [68, 30], [69, 27], [70, 28], [71, 23], [72, 25], [73, 21], [74, 24], [75, 22], [76, 23], [77, 22], [78, 25], [79, 22], [80, 22], [81, 25], [82, 25], [83, 23], [84, 19], [85, 19], [86, 19], [87, 21], [88, 23], [89, 25], [90, 24], [91, 26], [92, 24], [93, 27], [94, 24], [95, 27], [96, 29], [97, 29], [98, 29], [99, 30], [100, 30]

Then the following can be plotted.

```
> PLOT (CURVES ([res]), TITLE (''Plot of total number of
  skips on hire against day''), AXESLABELS (''i=day'',
  ''x=skips''), AXESSTYLE(BOX));
```



Note how the argument of 'CURVES' is a *list* (*res* has been enclosed in square brackets) of 100 points, each point expressed as a list of two values, its Cartesian coordinates. Now observe in the plot that the starting state $x = 0$ is hardly representative of a state chosen at random from the stationary distribution. However, by around day 10 the sequence $\{x_n\}$ appears to reflect near-stationary behaviour. Therefore, the remainder of the sequence will be used to estimate $E(X)$. First a sequence $x_{11}, \ldots, x_{100}$ needs to be constructed.

```
> data := seq (op (2, res[i]), i=11 .. 100);
```

data := 29, 30, 30, 28, 25, 30, 26, 30, 26, 23, 25, 25, 28, 30, 24, 24, 28, 24, 21, 22, 21, 20, 22, 25, 23, 29, 29, 29, 26, 25, 27, 27, 30, 30, 27, 30, 25, 23, 21, 16, 20, 20, 18, 22, 23, 26, 25, 29, 27, 26, 30, 27, 29, 27, 25, 26, 30, 30, 27, 28, 23, 25, 21, 24, 22, 23, 22, 25, 22, 22, 25, 25, 23, 19, 19, 19, 21, 23, 25, 24, 26, 24, 27, 24, 27, 29, 29, 29, 30, 30

Note how 'op' extracts the second element of *res*[*i*], and this is repeated for items 11 through 100 of the original sequence. Next the 'describe' function will be used within the statistics package of Maple to compute the mean of the *list*, [data].

```
> describe[mean] ([data]);
```

$$\frac{455}{18}$$

The result is expressed as a rational number. To obtain the result in decimal form the function 'evalf' is called. The '%' argument is the last computed value.

```
> evalf (%);
```

25.27777778

Now the number of skips available will be changed from 30 to 28. The next execution group will calculate the mean. Notice that there are colons at the end of the first two statements. These suppress the respective outputs and only the result for the mean will be seen. Also, note that in the execution group below there are no MAPLE prompts after the first line. This is achieved by using 'shift-return' rather than 'return'.

```
> res1 :=skip (5, 0.2, 28, 0, 100) :
  data1 :=seq (op (2, res1[i]), i=11 .. 100):
  evalf (describe [mean] ([data1]));
```

24.34444444

Suppose that the number of available skips is reduced further, this time from 28 to 26.

```
> res2 : =skip (5, 0.2, 26, 0, 100) :
  data2 : =seq (op (2, res2 [i]), i=11 .. 100) :
  evalf (described [mean] ([data2])) ;
```

23.44444444

It is pleasing that the *estimates* of the expected number of skips on hire, under stationarity, decreases as the availability is decreased from 30 to 28 to 26. This must be the case for the *actual* (as opposed to the estimated) expectations. However, it is not guaranteed for the estimates. Had *different seeds* been used for the three experiments there would be less likelihood of preserving the decreasing expectations in the estimates. At least by using the *same seed* in each case we have (inadvertently?) ensured a reasonably good experimental design that attempts to reduce the variance of the difference between any two of these estimates.

Now the experiment will be repeated for various K in order to find the optimal value for given cost parameters. Each realization is based upon a larger sample of 500 days, the burn-in time being selected as 10 days. The procedure '*skipprofit*' below will deliver a sequence $[K, pf[K]], K = 20, \ldots, 30$, where $pf[K]$ is the estimate of expected daily profit when the availability is K skips.

```
> skipprofit : =proc (c0, cf, cv, p) local K, res, datab,
                 avprofit;
   for K from 20 to 30 do:
      res: =skip (5, 0.2, K, 0, 500):
      datab: =seq (op (2, res[i]), i=10 .. 500):
      avprofit [K]: = [K, (p*cf+cv) *evalf(describe[mean]
                      ([datab])) −c0*K]:
   end do:
   seq (avprofit [K], K=20 .. 30);
   end proc;
```

```
skipprofit := proc(c0, cf, cv, p)
local K, res, datab, avprofit;
    for K from 20 to 30 do
        res := skip(5, 0.2, K, 0, 500);
        datab := seq (op(2, res[i]), i = 10 .. 500);
        avprofit[K] :=[K, (p*cf +cv)*evalf(describe[mean]([datab]))
                    −c0* K]
    end do;
    seq(avprofit[K], K = 20 .. 30)
end proc
```

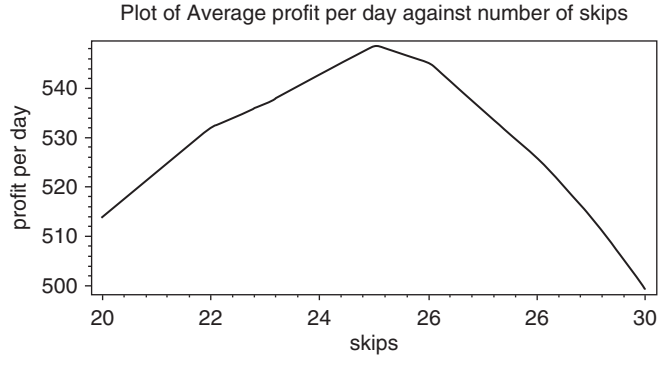
This will be called with a fixed cost of £40 per skip per day, a fixed rental price of £100 per hire, and a variable rental price of £50 per skip per day, together with a 0.2 probability of return each day.

```
> profit :=skipprofit (40, 100, 50, 0.2);
```

profit := [20, 513.747454], [21, 523.217922], [22, 531.975560], [23, 536.741344], [24, 542.647658], [25, 548.696538], [26, 545.193483], [27, 535.417516], [28, 525.926680], [29, 513.869654], [30, 499.246436]

Now these results will be plotted.

```
> PLOT (CURVES ([profit]), TITLE (''Plot of Average
  profit per day against number of skips''), AXESLABELS
  (''skips'', ''profit per day''), AXESSTYLE(BOX));
```



The estimated optimal strategy is to have 25 skips and the estimated long-run profit per day is approximately £549 per day. What sort of belief do we have in the proposed optimal 25 skips? This set of experiments should really be repeated with a different seed. The table below shows the seed, the optimal K, and the optimal profit for five sets of experiments in all. The optimal is $K = 25$ in three cases and $K = 24$ in two cases. Therefore, the optimal strategy is to select (with the aid of a single random number) 25 with probability 0.6 and 24 with probability 0.4. In the table below we should not be too concerned that the optimal profit shows considerable variations between experiments. In

Seed	Optimal K	Profit
5691443	25	548.7000
5691444	25	549.9800
21349109	24	536.6600
27111351	25	526.4600
3691254	24	503.8700

a decision problem such as this it is the *change* in the (unknown) expected profit resulting from a move away from the optimal decision that is important. A typical profit versus K plot (for example, the one shown above) reveals that there is a small reduction in profit and therefore a small cost associated with incorrectly choosing $K+1$ or $K-1$ rather than the optimal K.

1.5 Problems (see Appendix 1)

1. Use a Monte Carlo method, based upon 1000 random standard normal deviates, to find a 95 % confidence interval for $\int_{-\infty}^{\infty} \exp\left(-x^2\right) |\cos x| \, \mathrm{d}x$. Use the Maple 'with(stats)' command to load the stats package. The function 'stats[random,normald](1)' will generate a single random standard normal deviate.
2. Use a Monte Carlo method to find a 95% confidence interval for

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\left\{-\frac{1}{2}\left[x^2+(y-1)^2-\frac{x\,(y-1)}{10}\right]\right\} \mathrm{d}x \; \mathrm{d}y.$$

3. A machine tool is to be scrapped 4 years from now. The machine contains a part that has just been replaced. It has a life distribution with a time-to-failure density $f\,(x) = x\mathrm{e}^{-x}$ on support $(0, \infty)$. Management must decide upon one of two maintenance strategies. The first is to replace the part whenever it fails until the scrapping time. The second is to replace failures during the first two years and then to make a preventive replacement two years from now. Following this preventive replacement, the part is replaced on failures occurring during the second half of the 4 year span. Assume that replacements are instantaneous and cost £c_{f} on failure and £c_{p} on a preventive basis. Simulate 5000 realizations of 4 years for each policy and find a condition on $c_{\mathrm{p}}/c_{\mathrm{f}}$ for preventive replacement to be the preferred option.
4. Two points A and B are selected randomly in the unit square $[0, 1]^2$. Let D denote the distance between them. Using Monte Carlo:
   (a) Estimate $E(D)$ and $\mathrm{Var}(D)$.
   (b) Plot an empirical distribution function for D.
   (c) Suggest a more efficient method for estimating $P\,(D > 1.4)$, bearing in mind that this probability is very small.

5. An intoxicated beetle moves over a cardboard unit circle $x^2 + y^2 < 1$. The (x, y) plane is horizontal and the cardboard is suspended above a wide open jar of treacle. In the time interval $[t, t + \delta t)$ it moves by amounts $\delta x = Z_1 \sigma_1 \sqrt{\delta t}$ and $\delta y = Z_2 \sigma_2 \sqrt{\delta t}$ along the x and y axes where Z_1 and Z_2 are independent standard normal random variables and σ_1 and σ_2 are specified positive constants. The aim is to investigate the distribution of time until the beetle arrives in the treacle pot starting from the point (x_0, y_0) on the cardboard.

   (a) Write a procedure that simulates n independent times between starting at the point (x_0, y_0) and landing in the treacle. The function 'stats[random,normald](1)' creates a random standard normal deviate.

   (b) Plot a histogram showing the distribution of 200 such times when $\sigma_1 = \sigma_2 = 1$, $\delta t = 0.01$, and $x_0 = y_0 = 0$. To create a histogram, load the subpackage 'statplots' using 'with(statplots)' and use the function 'histogram(a)' where a is a *list* of the 200 times.

6. The following binomial model is frequently used to mimic share price movements. Let S_i denote the price at time ih where $i = 0, 1, 2, \ldots$ and h is a positive time increment. Let μ and σ denote the growth rate and volatility respectively. Let

$$u = \frac{1}{2}\left(e^{-\mu h} + e^{(\mu+\sigma^2)h}\right) + \frac{1}{2}\sqrt{\left(e^{-\mu h} + e^{(\mu+\sigma^2)h}\right)^2 - 4},$$
$$v = u^{-1},$$
$$p = \frac{e^{\mu h} - v}{u - v}.$$

   Then

$$S_i = X_i S_{i-1}$$

   where X_i, $i = 0, 1, \ldots,$ are independent Bernoulli random variables with distribution $P(X_i = u) = p$, $\text{Prob}(X_i = v) = 1 - p$ for all i.

   (a) Simulate the price at the end of each week during the next year when $S_0 = 100$ pence, $\mu = 0.2$ per annum, $\sigma = 0.3$ per annum, and $h = 1/52$ years.

   (b) Now suppose there are 252 trading days in a year. Put $h = 1/252$. For any realization let $S_{\max} = \max\left(S_j : j = 0, \ldots, 756\right)$. Let loss $= S_{\max} - S_{756}$. Loss denotes the difference between selling the share at its peak value during the next 3 years and selling it after 3 years. Simulate 200 realizations of loss and construct an empirical distribution function for it. You will need to sort the 200 values. Do this by loading the 'stats' package and using the function 'transform[statsort](x)' where x is a list of the data to be sorted. Note that if the order statistics for loss are $x_{(1)}, \ldots, x_{(n)}$ then an unbiased estimate of $P\left(X < x_{(i)}\right)$ is $i/(n+1)$.

7. Consider a single server queue. Let a_i denote the interarrival time between customer $i-1$ and customer i, s_i the service time of customer i, and w_i the waiting time in the queue (i.e. the time between arrival and start of service) for customer i.

(a) Show that $w_i = \max(0, w_{i-1} - a_i + s_{i-1})$.

(b) Now consider an M/M/1 queue (that is one in which arrivals follow a Poisson process and service durations are exponentially distributed) in which the arrival rate is λ and the service rate is μ. Write a procedure that simulates $w_1, \ldots, w_n$ given w_0, λ, and μ.

(c) Experiment with different values for the traffic intensity λ/μ plotting w_i against i to demonstrate queues that achieve stationary behaviour (i) quickly, (ii) slowly, and (iii) never. In cases (i) and (ii) provide point estimates of the expectation of w_i in the steady state.

2

Uniform random numbers

All simulations work on a 'raw material' of *random numbers*. A sequence $R_1, R_2, \ldots$ is said to be a random number sequence if $R_i \sim U(0, 1)$ for all i and R_i is independent of R_j for all $i \neq j$. Some authors use the phrase 'random numbers' to include variates sampled from any specified probability distribution. However, its use here will be reserved solely for $U(0, 1)$ variates.

How can such a sequence be generated? One approach is to use a physical randomizing device such as a machine that picks lottery numbers in the UK, a roulette wheel, or an electronic circuit that delivers 'random noise'. There are two disadvantages to this. Firstly, such devices are slow and do not interface naturally with a computer. Secondly, and paradoxically, there is often a need to *reproduce* the random number stream (making it nonrandom!). This need arises, for example, when we wish to control the input to a simulation for the purpose of verifying the correctness of the programming code. It is also required when we wish to compare the effect of two or more policies on a simulation model. By using the same random number stream(s) for each of the experiments we hope to reduce the variance of estimators of the *difference* in response between any two policies.

One way to make a random number stream reproducible is to copy it to a peripheral and to read the numbers as required. A peripheral could be the hard disc of a computer, a CD ROM, or simply a book. In fact, the RAND corporation published 'A million random digits with 100 000 random normal deviates' (Rand Corporation, 1955), which, perhaps, was not that year's best seller. Accessing a peripheral several thousands or perhaps millions of times can slow down a simulation. Therefore, the preferred approach is to generate *pseudo-random numbers* at run-time, using a specified deterministic recurrence equation on integers. This allows fast generation, eliminates the storage problem, and gives a reproducible sequence. However, great care is needed in selecting an appropriate recurrence, to make the sequence *appear* random.

Simulation and Monte Carlo: With applications in finance and MCMC J. S. Dagpunar
© 2007 John Wiley & Sons, Ltd

2.1 Linear congruential generators

These deliver a sequence of non-negative integers $\{X_i, i = 1, 2, \dots\}$ where

$$X_i = (aX_{i-1} + c) \bmod m \quad (i = 1, 2, \dots).$$

The recurrence uses four integer parameters set by the user. They are: $a(>0)$ a *multiplier*, $X_0(\geq 0)$ a *seed*, $c(\geq 0)$ an *increment*, and $m(>0)$ a *modulus*. The first three parameter values lie in the interval $[0, m-1]$. The modulo m process returns the remainder after dividing $aX_{i-1} + c$ by m. Therefore, $X_i \in [0, m-1]$ for all i. The pseudo-random number is delivered as $R_i = X_i/m$ and so $R_i \in [0, 1)$. The idea is that if m is large enough, the discrete values $\frac{0}{m}, \frac{1}{m}, \frac{2}{m}, \dots, \frac{m-1}{m}$ are so close together that R_i can be treated as a continuously distributed random variable. In practice we do not use $X_i = 0\,(R_i = 0)$ in order to avoid problems of division by zero and taking the logarithm of zero, etc. As an example consider the generator

$$X_i = (9X_{i-1} + 3) \bmod 2^4 \quad (i = 1, 2, \dots). \tag{2.1}$$

Choose $X_0 \in [0, 15]$, say $X_0 = 3$. Then $X_1 = 30 \bmod 2^4 = 14$, $X_2 = 129 \bmod 2^4 = 1, \dots$. The following sequence for $\{R_i\}$ is obtained:

$$\frac{3}{16}, \frac{14}{16}, \frac{1}{16}, \frac{12}{16}, \frac{15}{16}, \frac{10}{16}, \frac{13}{16}, \frac{8}{16}, \frac{11}{16}, \frac{6}{16}, \frac{9}{16}, \frac{4}{16}, \frac{7}{16}, \frac{2}{16}, \frac{5}{16}, \frac{0}{16}, \frac{3}{16}, \dots. \tag{2.2}$$

The *period* of a generator is the smallest integer λ such that $X_\lambda = X_0$. Here the sequence repeats itself on the seventeenth number and so $\lambda = 16$. Clearly, we wish to make the period as large as possible to avoid the possibility of reusing random numbers. Since the period cannot exceed m, the modulus is often chosen to be close to the largest representable integer on the computer.

There are some number theory results (Hull and Dobell, 1962) that assist in the choice of a, c, and m. A *full period* $(=m)$ is obtained if and only if

c and m *are relatively prime* (*greatest common divisor of* c and m is 1); (2.3)

$a-1$ is a multiple of q for every prime factor q of m; (2.4)

$a-1$ is a multiple of 4 if m is. (2.5)

Linear congruential generators can be classified into *mixed* $(c > 0)$ and *multiplicative* $(c = 0)$ types.

2.1.1 Mixed linear congruential generators

In this case $c > 0$. A good choice is $m = 2^b$ where b is the maximum number of bits used to represent positive integers with a particular computer/language combination. For example, many computers have a 32 bit word for integers. One bit may be reserved for the sign, leaving $b = 31$. Such a computer can store integers in the interval $\left[-2^{31}, 2^{31} - 1\right]$.

By choosing $m = 2^b$ the generator will have a full period ($\lambda = m$) when c is chosen to be odd-valued, satisfying condition (2.3), and $a - 1$ to be a multiple of 4, satisfying conditions (2.4) and (2.5). This explains why the generator (2.1) is a full period one with $\lambda = 16$.

In Maple there is no danger that $aX_{i-1} + c$ exceeds N, the largest positive integer that can be stored on the computer. This is because Maple performs arithmetic in which the number of digits in N is essentially limited only by the memory available to Maple. The Maple procedure 'r1' below uses parameter values $m = 2^{31}$, $a = 906185749$, $c = 1$. Note the importance of declaring 'seed' as a global variable, so that the value of this variable can be transmitted to and from the procedure. The period of the generator is $m = 2147483648$, which is adequate for most simulations. This generator has been shown to have good statistical properties (Borosh and Niederreiter, 1983). More will be stated about statistical properties in Sections 2.2 and 2.4.

```
> r1 := proc() global seed;
  seed:= (906185749*seed + 1) mod 2^31;
  evalf(seed/2^31);
  end proc:
```

The code below invokes the procedure five times starting with seed = 3456:

```
> seed:= 3456;
  for j from 1 to 5 do;
  r1();
  end do;
```

```
seed:= 3456
.3477510815
.2143113120
.7410933147
.4770359378
.6231261701
```

The generator took about 12 microseconds per random number using a Pentium M 730 processor. The generator 'r2' below (L'Ecuyer, 1999) has a far more impressive period. The parameter values are $m = 2^{64}$, $a = 2,862,933,555,777,941,757$, $c = 1$. The period is 18, 446, 744, 073, 709, 551, 616 and the execution speed is not much slower at 16 microseconds per random number.

```
> r2:= proc() global seed;
  seed:= (2862933555777941757*seed + 1) mod 2^64;
  evalf(seed/2^64);
  end proc:
```

In most scientific languages (e.g. FORTRAN90, C++) the finite value of N is an issue. For example, N may be $2^{31} - 1$ when the word size is 32 bits. Explicit calculation of $aX_{i-1} + c$ may cause a fatal error through *overflow* when $aX_{i-1} + c \notin \left[-2^{31}, 2^{31} - 1\right]$. However, in some implementations, the integer following N is stored as $-(N+1)$